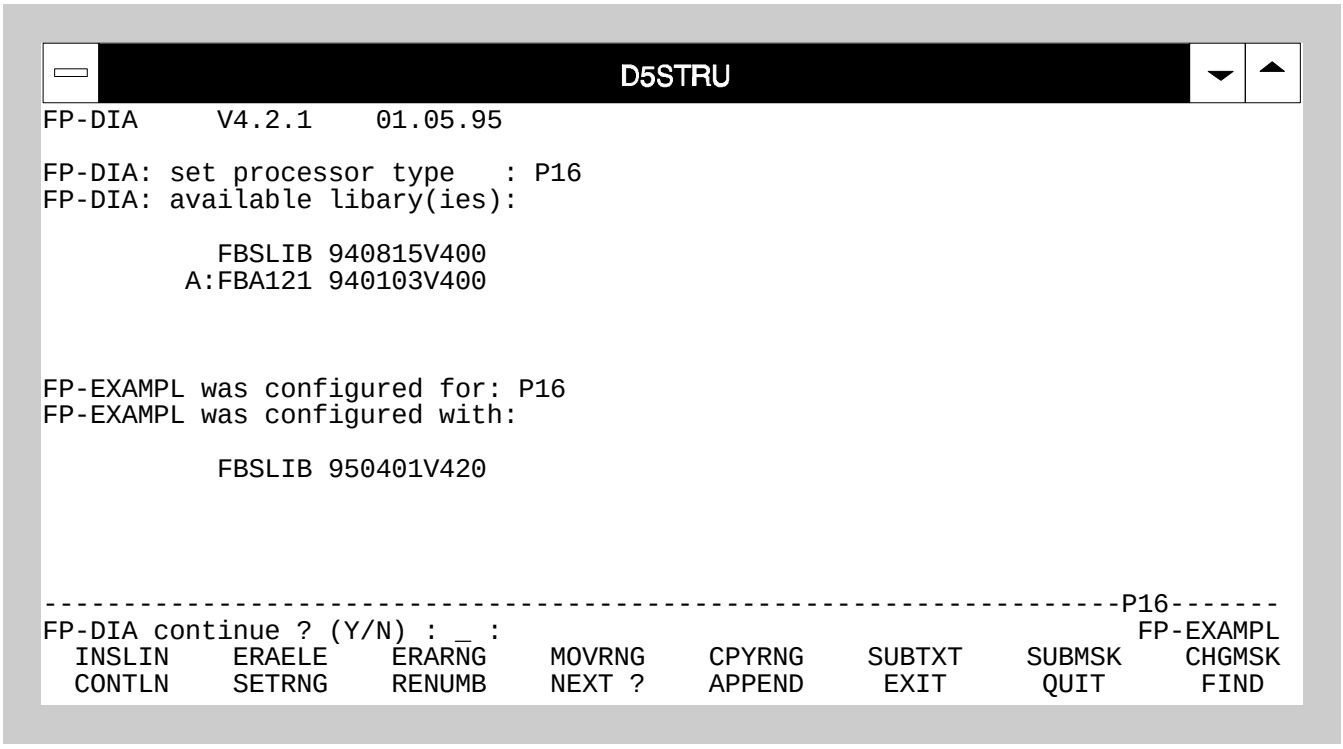
1 Editor, general

Contents

1.1 Calling-up the editor

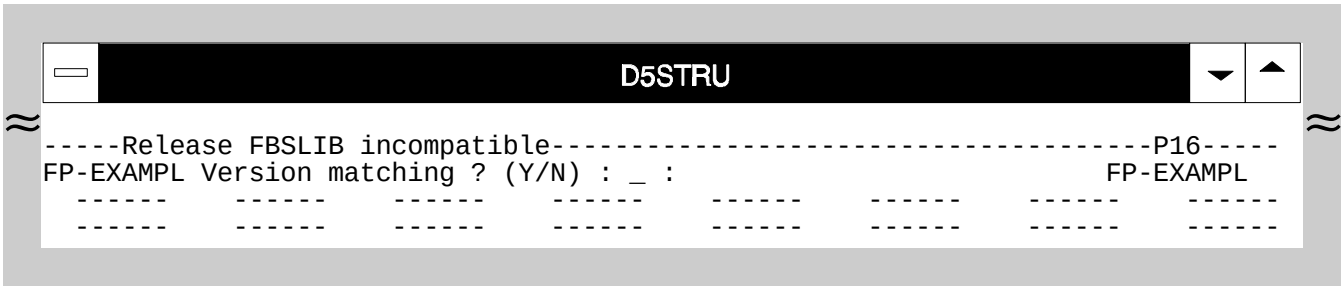
The editor is called-up from the menu bar of the basic dialog (MP Edit, FP Edit). When calling-up, for example, a function package is clicked-on in the scrolling processor program menu, and then selected with **FP Edit / Configure**. The "D5 STRU" editor window is then displayed.

The available libraries are displayed, and a prompt appears (3rd line from below), as to whether the dialog should be continued.



Those block libraries which are available for the selected processor type (P16, P32) are displayed, and which processors are used in the function package. You are prompted as to whether processing should continue. The FP is displayed when "Y" is entered. FP-DIA is aborted and the system returns to the basic dialog when an "N" is entered.

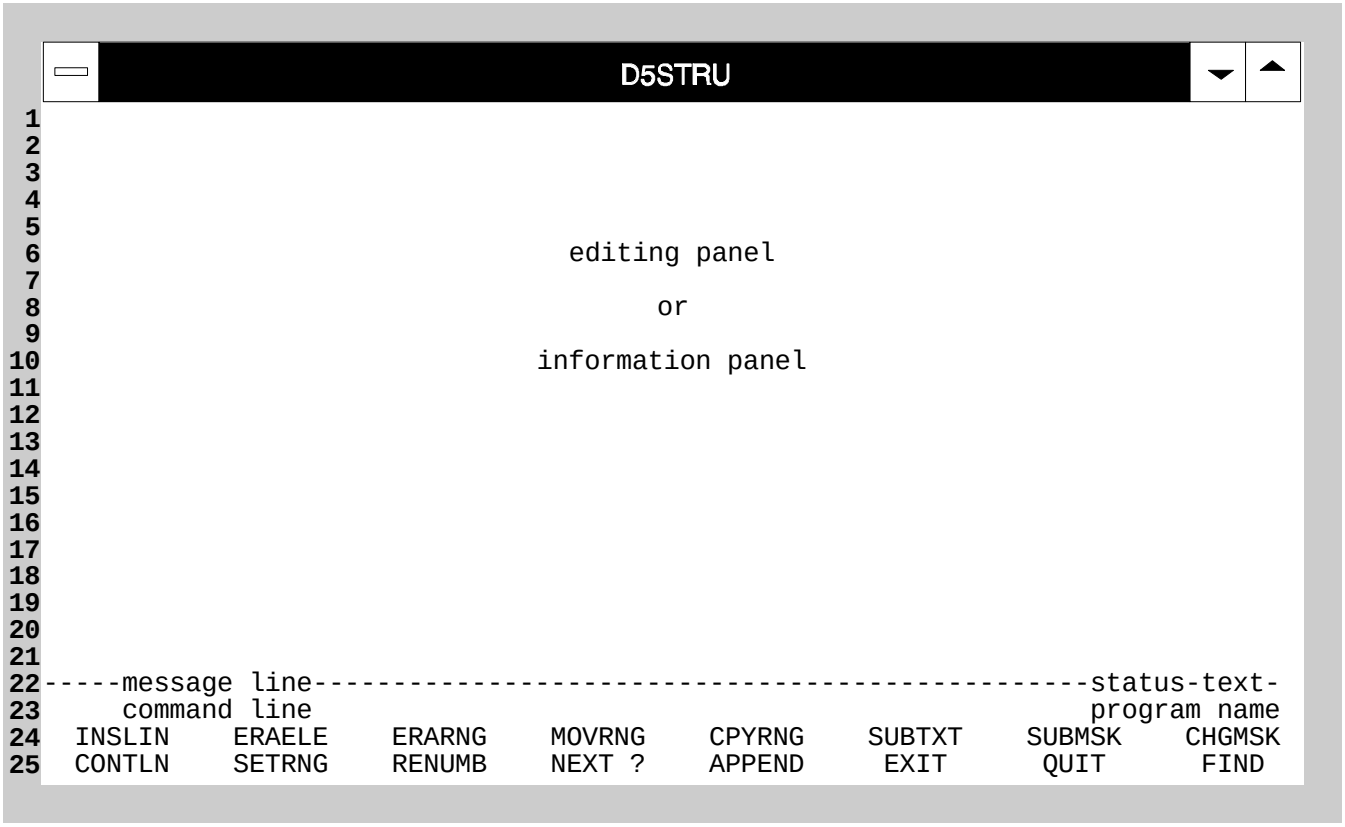
If the libraries were, in the meantime, updated (version change), then the following message and question are displayed:



After a "Y" has been entered, dialog is continued with the question "FP-DIA continue ? (Y/N)". If an "N" is entered, the function package can only be read, and not processed.

The same procedure is executed when the master program editor is called-up (MP Edit). The MP is handled instead of the FP, and the IB libraries are displayed instead of the FB libraries.

1.2 Editor window structure



The display has 25 lines, each with 80 columns, whereby, not all line- and column positions can be written into by the user.

For interactive dialog, the first 21 lines (working range), are used as editor field for direct entries and changes. The first four columns are always reserved for the line number.

Display lines 22 ... 25 are used as command- and message field. Fault messages are displayed in line 22, starting from the left. Status messages are possible at the righthand edge of the line, e.g., the "insert character" is displayed with INSERT. The Ins (insert/overwrite mode) is used to toggle between the overwrite- and insert modes. All of these messages appear inversely (dark on a bright background). Line 22 is also used as boundary line between the editor and command field.

The following line, line 23, is the command line. Questions are displayed here regarding further program processing in plain text, or supplementary information can be provided for selected function keys. The current program name (FP- or MP name) is continually displayed as status message at the end of the command line.

The remaining part of the command field includes the function keys. The function key assignment is different depending on the configuring step. It is possible to toggle between the editor field and function keys by depressing the Esc key. From V4.2 on the function keys can also be invoked directly. Example: F7 invokes SUBMSK, <shift> F7 invokes QUIT.

A detailed description of the function keys is provided in the sections handling the particular configuring step. The significance of the function keys for the editor (FP- and MP editor) are shown in the next section.

1.3 Calling-up blocks

An existing program (function package, master program) can be changed using the editor. Blocks (function blocks, modules), which are stored in existing block libraries, depending on the configuring level, can be called-up. The cursor must be positioned at the required program line, and the block name and type code should be entered as follows:

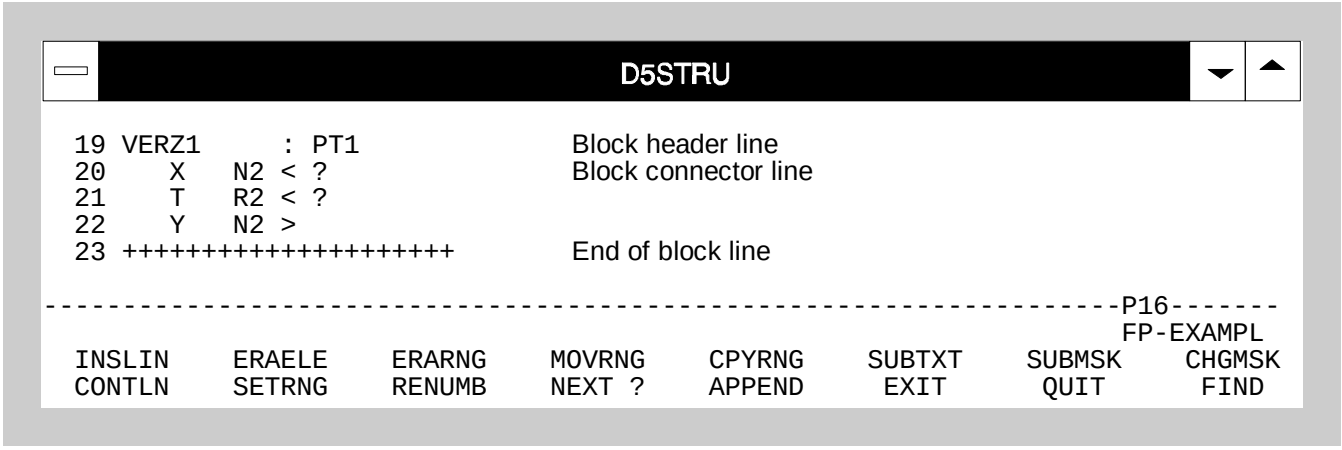
```
block name:block type  Enter  (Enter key)
```

Block name: Names selected by the user; max. 6 characters; all upper case letters of the international alphabet can be used, digits 0...9 and underline; the first character must be a letter

Block type: Type name from the block library (block catalog)

Example: `VERZ1:PT1` Enter (enter key)

An empty mask is then displayed which must be completed. Example of an empty mask (PT1 delay element):



The cursor must be positioned at the required program line, and the

If a Macro shall be invoked, block name and type code should be entered as follows:

```
makroname:makrotyp.M  Enter  (Eingabetaste)
```

The Macro type names of the invokable Macros are determined with the function **Options / Configurable Macros**.

Mask lines, with a question mark "?" must be completed, whereby the question mark is then overwritten. If the question mark is kept, this means that this line is still not complete. The line is, in this case, not checked for syntax errors. The user can mark any line with a question mark. The number of incomplete lines (lines with question mark (?)) is displayed when the editor is terminated. Example of an incomplete line.:

```
  21    R2 < 12 or 22[ms] ?
```

Function packages and master programs with incomplete lines cannot be compiled error-free.

1.4 Cursor movement

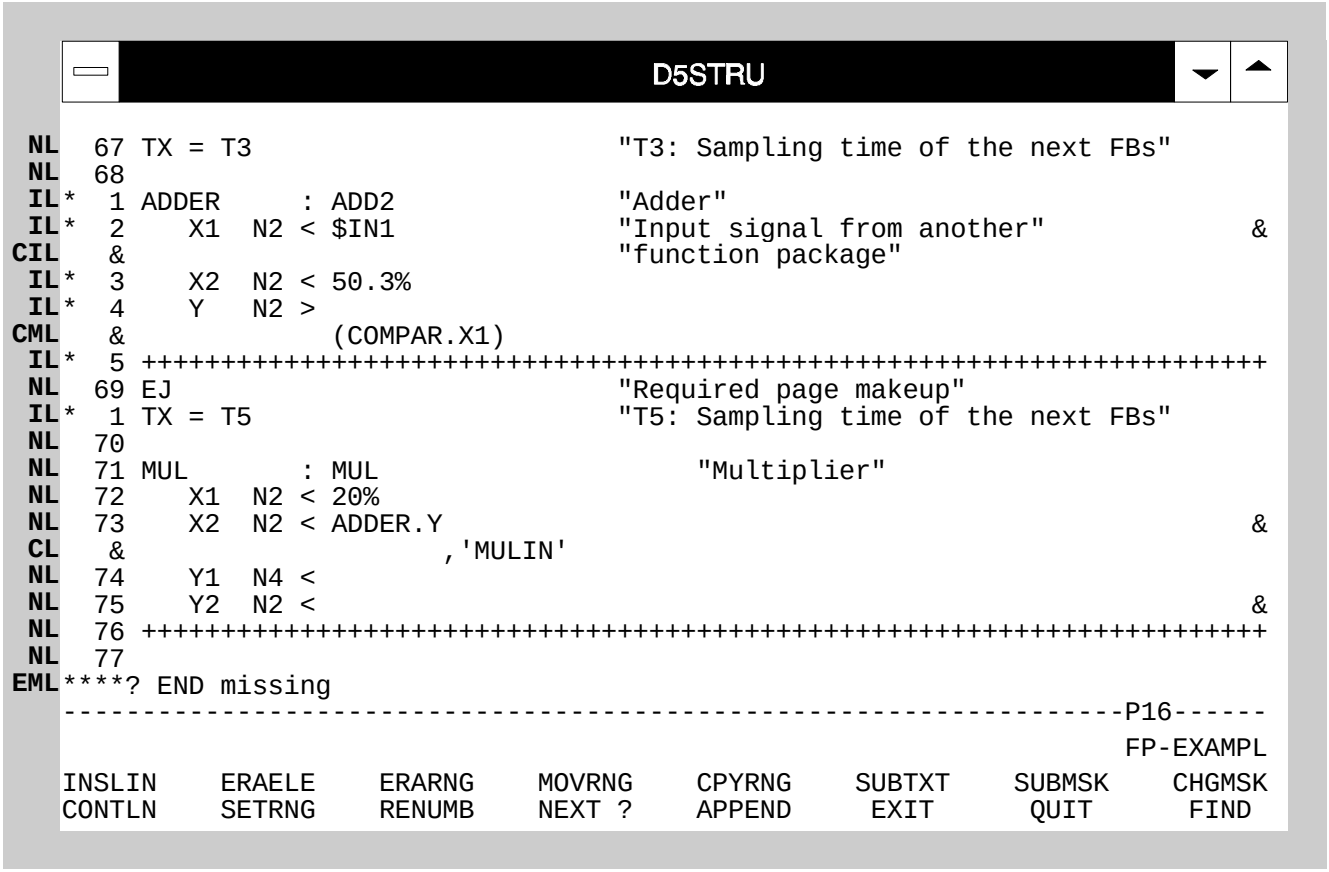
The cursor can only be moved in fields where entries can be made.

- ↓ : Move cursor one line down
- ↑ : Move cursor one line up
- → : Move cursor one column to the right
- ← : Move cursor one column to the left
- Enter : Move the cursor to the start or to the first input field of the next line. When this key is depressed in the last line, a new line is generated.

1.5 Line sub-division and numbering

If a mask is attached at the end of the program, the program lines are consecutively numbered. If a mask is inserted between already existing blocks, the new program lines are designated with a star (*), and re-numbered starting from 1. Thus, the old line numbers are retained.

The following example shows a program section with the various STRUC L line numbers and line types:



IL: Insert line

CIL: Continuation line of the inserted line

CL: Continuation line (only in connector lines)

EML: Error message line

CML: Compiler message line (cross-reference)

NL: Standard line

The first four columns of a line are reserved for the line number on the screen and in the STRUC L documentation. The dialog program automatically numbers a program while it is being generated (standard lines).

¨ **Continuation line**

Data can be entered in any line up to column 79. If longer entries are necessary or if entries are to be transparently configured, continuation lines can be opened in the connector lines. This is realized by depressing the function key CONTLN (Esc ¯ CONTLN Enter). An and symbol (&) then appears at the end of the current line (column 80), and the new continuation line is then designated with this character. Several continuation lines can be entered one after the other. If an unwritten continuation line is terminated with Enter (enter key), then it is deleted.

¨ **Compiler supplements**

The compiler enters cross references to input connectors, with which the appropriate output is connected, in the compiler message lines at the function block outputs. These lines cannot be deleted. Further, the compiler error message lines with the designation ****?, are generated by the compiler, which can be deleted at any time using the ERAELE function.

¨ **Insertion line**

If lines are inserted using the function INSLIN, then these are designated with a star (*), and are re-numbered starting with "1". Thus, insertion has no effect on the remaining line numbers, and old list line numbers can still be used. However, when inserting between insertion lines, these are re-numbered.

Two pieces of information may be required to clearly identify lines (e.g. 15 * 3):

- the last standard line number (15)
- the required insertion line number (*3)

If lines are deleted in interactive dialog, the insertion line numbers are re-numbered; standard line numbers are not re-numbered.

The complete program can be re-numbered using the function key RENUMB. Continuation lines, compiler message lines and error message lines are in this case not changed.

1.6 Comments

Comments can be entered for documentation and explanation. Header masks and module masks of the master program already include comments.

Comments must be within quotation marks (" "). If a comment text takes up several lines, the comment component of each line must start and end with quotation marks.

It is recommended that comments are started from a specific character position. The STRUC L editor supports this, as the cursor is always positioned to character position 41 using the tab key (⇥).

Comments are not taken into account during compilation, and are therefore no longer available after reverse documentation from the EPROM memory module of the processor board.

1.7 Error identification and error messages

1.7.1 Syntax check

A line syntax check is executed when editing, when a line or a continuation line area has been left.

The following are identified by the syntax check, for example:

- conflicting dimension information
- missing quotation marks for comments
- illegal characters within names

Logical errors caused by conceptual faults and errors, which are only obvious in conjunction with other lines or programs, cannot be identified by the line check.

The error messages are displayed in the message line, and the cursor is positioned in the text, at the first position of the field with the erronous input, e.g.:

```
---- Dimension definition missing ------------
```

A line can only be exited, if the syntax is perfect. If the syntax check is not executed, a question mark (?) can be entered in the appropriate line. This question mark must however be located before the start of a comment (") or string ('). For continuation lines, the question mark should be entered in the associated standard line. If all of the question marks have not been removed before compilation, the compilation run will be aborted after pass 0 when a function package is being compiled (refer to Section 3.2).

1.7.2 Additional error messages

Error messages are also issued when illegal keys are depressed and when illegal entries are made in the command line.

Further, system errors can occur. These errors are described in more detail in the Appendix (Section 5). For some of these errors, it may be necessary to involve a system expert, who should then be informed of the displayed error code. Further, it might be necessary to provide the user program (FP,MP), which shouldn't have been able to have been changed after the system error message.

1.8 Editing within a line

For editing within a line, in addition to the alphanumeric keys, the cursor keys (® ¬) can be used. The entered characters can be overwritten, or deleted using the keys ← (delete character to the left of the cursor and Del (delete character at the cursor position). These editing capabilities are also valid for the command line.

Further, it is possible to toggle between the insert- and overwrite mode using the key 0 Ins (insert/overwrite mode).

1.9 Edit commands

When editing on the screen, the assignment of the commands to the function keys can be identified in the command field.

Commands with supplementary information, are assigned to the function keys of the upper command line, and must always be terminated with Enter ↵ .

Commands without supplementary information are executed using the function keys of the lower command line.

After the command has been executed, the cursor normally appears at the same display position which it previously had, and when a block is inserted, the cursor is positioned at the start of the new block.

When searching a text, the cursor is positioned directly after the searched for text, or in an input field of the line, if the text is located in the mask section.

The function keys are assigned as follows for editing:

```
INSLIN    ERAELE    ERARNG    MOVRNG    CPYRNG    SUBTXT    SUBMSK    CHGMSK
CONTLN    SETRNG    RENUMB    NEXT ?    APPEND     EXIT      QUIT      FIND
```

The significance of the individual function keys is described in the following pages. The following definitions are valid for the commands used there:

bst-name	Block name (name assigned by the configuring engineer)
bst type	Block type name (block name in the library)
Lower case letters	Identifying variables, expressions, parameters
▢	Key functions
| |	Obligatory input of the various possibilities to be selected
{ }	Comment not to be entered
Enter	Command completion key
_____	Mask components
=====	Key function, and at the same time mask component or status message

1.9.1 Commands with supplementary information

These commands require supplementary information:

Position information:

- line number
- block name
- special items $ T, $ E, $ B, $ F

Range information:

- $ R, $ A

If no supplementary information is entered, the current cursor position is assumed to be the position information.

Line number = normal line number (e.g. 67) or normal line number * insertion line number (e.g. 67*5)

INSLIN insert line

```
            |  {line pos. cursor}  |
            |  line number         |
INSLIN    : |  block name          | : Enter
======    -                        -
```

A blank line is inserted in the text using this command:

- in front of the current cursor line position
- in front of the line number
- in front of the specified block

Inserted lines are identified with a star (*), and re-numbered starting with "1".

It is not possible to insert within a mask.

Examples:
```
INSLIN  :           : Enter
======  -           -

INSLIN  : 20        : Enter
======  -           -

INSLIN  : 22*3      : Enter
======  -           -

INSLIN  : FBXYZ     : Enter
======  -           -
```

ERAELE erase element

```
            |  {line pos. cursor}  |
            |  line number         |
ERAELE    : |  block name          |: Enter
======    -                        -
```

This command allows individual lines to be deleted if they are not part of a block. The line, in which the cursor is located, or which is specified by the line number, is deleted. If the line is the header line of a block, the complete block is deleted. It is also possible to directly enter the name of a block.

Inserted lines are re-numbered.

Error message lines ****?, can also be deleted within a block.

Examples:
```
ERAELE :           : Enter
====== -           -

ERAELE : 20        : Enter
====== -           -

ERAELE : 42*2      : Enter
====== -           -

ERAELE : FBXYZ     : Enter
====== -           -
```

ERARNG erase range

```
              |  {range marked by SETRNG}           |
              | |line No. start  |   |line No. end  | |
ERARNG      : | |block name start| , |block name end| | : Enter
======      -                     -                   -
```

The ERARNG command deletes the range marked by SETRNG, without any further information. However, a range can also be directly specified using a line number or block name for start and end of the range to be deleted. The final line or the end block is also deleted.

Block components cannot be deleted.

Examples:
```
ERARNG : FBABC      , FBXYZ     : Enter
======   -          -           -

ERARNG : FBABC      , 63        : Enter
======   -          -           -

ERARNG :            ,           : Enter
======   -          -           -
```

MOVRNG move range

```
              | {marked range}                      |   | {line pos. cur}  |
              ||line No. strt.  |   |line No. end  ||   | line No. dest.   |
MOVRNG      : ||block name strt.| , |block name end|| , | block name dest. | : Enter
======      -                     -                  -                    -
```

MOVRNG allows complete areas and ranges to be shifted. The area can be defined using line numbers or block names. The range shifted always includes the final line or the final block. If the range is terminated with a block, and if the line number information is used, then the block header line must always be specified as end line.

If no information is provided in the command line regarding start and end of range, the range, marked with SETRNG, is shifted. The range is inserted in front of the actual line cursor position, in front of the the destination line number or in front of the destination block, and provided with insertion line numbers.

Parts of blocks cannot be shifted.

Examples:
```
MOVRNG : 20*2       , 20*12     ,           : Enter
======   -          -           -           -

MOVRNG : FBABC      , FBXYZ     , 24        : Enter
======   -          -           -           -
```

CPYRNG copy range

```
            | {marked range}                        |   | {line pos. cur}      |
            ||line No. start  |   |line No. end  ||   | line No. destination |
CPYRNG    : ||block name start| , |block name end|| , | block name dest.     | :  Enter
======    -                     -                      -                        -
```

Complete ranges can be copied using CPYRNG. The ranges can be defined using line numbers or block names. The range copied always includes the final line or the final block. If the range is terminated with a block, and if the line number information is used, then the final block header line must be specified.

If no information is provided in the command regarding start and end of range, the range, designated with SETRNG, is copied. The range is inserted in front of the current line cursor position or in front of the target line number or in front of the destination block, and provided with instruction line numbers.

Parts of blocks cannot be copied.

Examples:
```
CPYRNG : 20*2      , 20*12     ,          :  Enter
====== -           -           -          -

CPYRNG : FBABC     , FBXYZ     , 24       :  Enter
====== -           -           -          -
```

SUBTXT substitute text

```
                                  | {from cursor pos.} |
                                  |     $A             |
SUBTXT    : |'text1'| , |'text2'| , |     $R             | :  Enter
======    -           -           -                      -
```

Using this command, character sequences can be replaced by other character sequences. Generally, the first found character sequence is replaced from the current cursor position. By specifying $A, a search is made starting from the program start, and by specifying $R, in the range marked by SETRNG.

Parts of masks cannot be changed. New character sequences can only be entered in legal input fields, that means, that it may be necessary to shorten the character sequence.

```
SUBTXT    :          ,          ,          :  Enter
======    -          -          -          -
```

If a SUBTXT command is immediately followed by an additional SUBTXT command, without any additional information, then the last executed SUBTXT command is repeated, without taking into account $A and $R supplementary information. Thus, it is easy to replace a specific character sequence in a complete program.

Examples:
```
SUBTXT : 'Binary output','output'      ,     :  Enter
====== -                 -             -     -

SUBTXT : 'DIGOUT'       ,'OUT1'        , $A  :  Enter
====== -                 -             -     -

SUBTXT :                 ,             ,     :  Enter
====== -                 -             -     -
```

SUBMSK substitute mask

```
                                | block type   |
            | {line pos. cur} |   | block type.A |
            |  block name     |   | block type.B |
SUBMSK    : |  line No.       | , | block type.C | :  Enter
======    -                     -                  -
```

This command allows block masks to be replaced (master program, function package). In this case the data and comments, configured in the input fields of the existing block are transferred into the new mask. The block name assigned by the user is retained.

The block mask to be replaced is marked by the current cursor position or by the line number, whereby, both must refer to the header line of a block. However, a block name can also be explicitly specified.

Data and comments from the block to be replaced are transferred in the new mask, connector-related, i.e. connector data of the old mask are written into the connector of the new mask, which carries the same connector designator and type. If such a connector is not available these data are eliminated. For additional new connectors, the pre-assignment is entered from the mask library.

Examples:
```
SUBMSK :           , ADD2      :  Enter
======  -          -           -

SUBMSK : 20        , ADD2      :  Enter
======  -          -           -

SUBMSK : FBAC      , ADD2      :  Enter
======  -          -           -
```

CHGMSK change mask

```
                                | block type   |
            | {line pos.cur} |  | block type.A |
            |  block name    |  | block type.B |
CHGMSK    : |  line No.      | , | block type.C | :  Enter
======    -                    -                  -
```

This command permits block masks to be replaced in the function packages which were processed with the function Version / FP Transformation P16 ↔ P32, and include marked blocks. It is just like the SUBMSK command, however, with the difference that connector data of the old mask are written at the connector of the new mask, only if the connector designator and the data format (e.g. N) are identical (e.g. replacing an ADD2 block (N2 input) with an ADD24 block (N4 inputs).

Examples:
```
CHGMSK :           , ADD24     :  Enter
======  -          -           -

CHGMSK : 20        , ADD24     :  Enter
======  -          -           -

CHGMSK : FBAC      , ADD24     :  Enter
======  -          -           -
```

FIND find

```
            |   line No.    |   | {from cursor pos. } |
            |   block name  |   |     $A              |
            |    'text'     | , |     $R              |
FIND      : |$(jump mark  )| , |     $R              | : Enter
====      -                   -                       -
```

This command is used to search lines, texts and blocks. The search is started from the current cursor position, when $A is specified, a search is made from the program start, and for $R, in the range marked using SETRNG. The cursor positions itself at the first found line. For texts, it is positioned after the last character.

Important program positions can be found as follows:

```
            | $T |                          cursor at the program start (top)
            | $E |                          cursor at the program end (end)
            | $B |                          cursor at the start of a marked range (begin)
FIND      : | $F | ,       : Enter          cursor. at the end of a marked range (finish)
====      -       ,       -

FIND      :       ,       : Enter
====      -       -       -
```

If a FIND command is directly followed by an additional command without any further information, the last executed FIND command is repeated, without taking into account the $A and $R supplementary information. First, it is easy to find all program positions, where a specific text is located.

Examples:

```
FIND : 20*4                    ,      : Enter
==== :                         -      -

FIND : 'comment'               , $A   : Enter
==== -                         -      -

FIND :                         ,      : Enter
==== -                         -      -

FIND : $E                      ,      : Enter
==== -                         -      -
```

1.9.2 Commands without supplementary information

CONTLN continuation line

CONTLN
======

Continuation lines can be generated using this key. An AND symbol (&) is then displayed at the end of the current line and the new continuation line is designated with an AND (&).

SETRNG set range

SETRNG
======

When the key is first depressed, a range flag is first set for the line, in which the cursor is positioned. When the key is depressed again, the range betwen the instantaneous cursor position (line) and the line first selected when the key was first depressed, are marked as range. The same is true when the key is again depressed.

If the cursor is positioned at the start line of a marked range, and if the key is depressed in this status, then the range marking is deleted.

A marked range can be shifted, copied or deleted using the MOVRNG, CPYRNG and ERARNG functions. It is also possible to refer to such a range using the search (FIND) and replace functions (SUBTXT).

RENUMB renumber

RENUMB
======

All lines, including the insertion lines are re-numbered. The cursor remains at the the current position

NEXT ? next ?

NEXT ?
======

The cursor jumps into the next line, designated with a question mark, behind the actual cursor position. The cursor is then positioned at the question mark in the line. The line can either be an incomplete mask line, an error message line from the compiler or a line with question marks in the comments field

APPEND append

```
APPEND    fp-name:        :  or
======    mp-name:        :
```

This function allows a function package or master program to be attached to the end of the function package or master program presently in the memory.

The name of the function package or master program which is to be attached must be specified. The header data of the attached function package or master program, are removed as this can only be present once. This function allows data to be transferred from one function package or master program to another.

EXIT exit

```
EXIT
====
```

After calling-up, the following prompt is displayed: `archive/save ? (A/S) :  :`

If "A" is entered, data is archived and interactive dialog terminated.

If 'S' is entered data is saved, but interactive dialog can still be continued. This function allows data back-up without having to exit the interactive dialog system.

The system returns to the basic dialog after ending (with "A"). The number of incomplete lines with question marks (?) is displayed in the message line. A subsequent compiler run will be aborted if incomplete lines are present.

If the name of the current file was changed in the FB- or MP header, and if a file already exists with the newly selected names, the following message appears `'already available, overwrite ? (Y/N)'`. If a 'Y' is entered, the existing file is overwritten; if an 'N' is entered, the cursor jumps to the input field for the file names, where a new file name can be entered.

QUIT quit

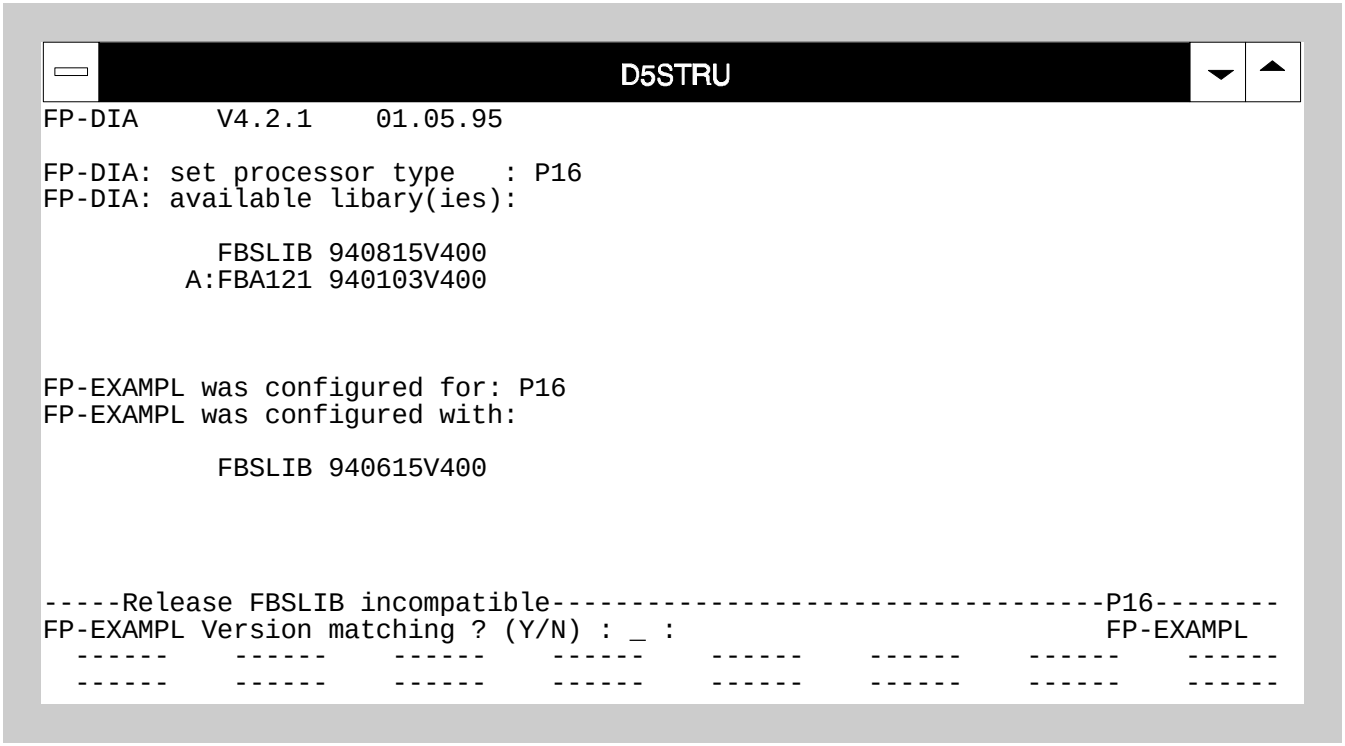
```
QUIT
====
```

The text file is exited without saving using this command. In order to prevent erronous application, the system prompts `'xx-DIA  abort without saving ? (Y/N)'` and only executes the command if a 'Y' is entered.
If a command with supplementary information was previously entered, which is not to be executed, then the command line can be exited using QUIT, without the command being processed. Thus, the QUIT key has two different functions.

1.10 Version adaption

If a change is made to a new version, the existing master programs and function packages must be adapted to this version. A version adaption is only possible if the system version is higher or the same as the version which is entered in the MP or FP. Otherwise, adaption is not possible, i.e., STRUC L is upwards- but not downwards compatible.

If version adaption is permissible, the interactive dialog system asks whether such as adaption is required. The window is as follows:



If a 'Y' is entered, the interactive dialog system enters the new version into the program, and compares all program blocks with the appropriate blocks in the mask library. In this case, the following incompatibilities could occur:

- blocks are no longer available in the library

The deleted blocks are marked with a question mark (?) in the header line.

e.g.: VERZ1 :?PT1

Adaption: Changing the software

- Changing the visible block masks
  These blocks are also marked with question marks (?).

  Adaption: Replacing the block mask using the SUBMSK function or new configuring

- Changing the invisible block parameters
  The changed parameters (e.g. computation time) are automatically replaced by the system